EPM–RT–2010-12

DÉTECTION ET CORRECTION AUTOMATIQUE DES DÉFAUTS DE CONCEPTION AU MOYEN DE L'APPRENTISSAGE AUTOMATIQUE POUR L'AMÉLIORATION DE LA QUALITÉ DES SYSTÈMES

Abdou Maïga
Département de Génie informatique et génie logiciel
École Polytechnique de Montréal

Septembre 2010

Poly

ÉCOLE POLYTECHNIQUE MONTRÉAL

EPM-RT-2010-12

Détection et correction automatique des défauts de conception au moyen de l'apprentissage automatique pour l'amélioration de la qualité des systèmes

Abdou Maïga
Département de génie informatique et génie logiciel
École Polytechnique de Montréal

Septembre 2010

©2010
Abdou Maïga
Tous droits réservés

Dépôt légal :
Bibliothèque nationale du Québec, 2010
Bibliothèque nationale du Canada, 2010

EPM-RT-2010-12
Détection et correction automatique des défauts de conception au moyen de l'apprentissage automatique pour l'amélioration de la qualité des systèmes
par : Abdou Maïga
Département de génie informatique et génie logiciel
École Polytechnique de Montréal

Toute reproduction de ce document à des fins d'étude personnelle ou de recherche est autorisée à la condition que la citation ci-dessus y soit mentionnée.

Tout autre usage doit faire l'objet d'une autorisation écrite des auteurs. Les demandes peuvent être adressées directement aux auteurs (consulter le bottin sur le site http://www.polymtl.ca/) ou par l'entremise de la Bibliothèque :

École Polytechnique de Montréal
Bibliothèque – Service de fourniture de documents
Case postale 6079, Succursale «Centre-Ville»
Montréal (Québec)
Canada H3C 3A7

Téléphone : (514) 340-4846
Télécopie : (514) 340-4026
Courrier électronique : biblio.sfd@courriel.polymtl.ca

Ce rapport technique peut-être repéré par auteur et par titre dans le catalogue de la Bibliothèque :
http://www.polymtl.ca/biblio/catalogue.htm

Université de Montréal

Détection et correction automatique des défauts de conception au moyen de l'apprentissage automatique pour l'amélioration de la qualité des systèmes

par
Abdou Maiga

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Rapport pour la partie orale
de l'examen pré-doctoral

Aout, 2010

© Abdou Maiga, 2010.

Université de Montréal
Faculté des études supérieures

Cet examen pré-doctoral intitulé:

Détection et correction automatique des défauts de conception au moyen de l'apprentissage automatique pour l'amélioration de la qualité des systèmes

présenté par:

Abdou Maiga

a été évalué par un jury composé des personnes suivantes:

Pascal Vincent,	président-rapporteur
Esma Aïmeur,	directeur de recherche
Yann-Gaël Guéhéneuc,	codirecteur
Bruno Dufour,	membre du jury

Examen accepté le: . . . . . .

RÉSUMÉ

La maintenance logicielle apparait comme l'activité la plus coûteuse dans le cycle du développement : plus de 80% des ressources lui sont consacrées. Au cours des activités de maintenance, l'architecture et la conception du logiciel sont très peu prises en compte. Il s'en suit une dégradation progressive de ces artefacts dus à des défauts de conception. Ces défauts peuvent avoir été introduits dès la première conception mais également par les maintenances du logiciel. La dégradation de la conception du logiciel rend encore plus difficile la compréhension du logiciel et les maintenances à venir, créant ainsi un cycle vicieux. Nous nous proposons dans ce projet de recherche de contribuer à réduire la dégradation des conceptions logicielles en mettant en place un système intégré de détection et de correction automatiques des défauts de conception et également un suivi de la qualité de la conception. Ce système, nommé SUDERCO, est basé sur l'apprentissage automatique et vise à fournir un cadre souple et évolutif pour aider à réduire les coûts de maintenance par la préservation de la conception.

Mots clés: défauts de conception, SVM, système de recommandation, détection, correction, refactorisations, Java.

ABSTRACT

Software maintenance is emerging as the most expensive activity in the development cycle: more than 80% of resources are devoted to it. During maintenance activities, architecture and design of the software are rarely taken into account. It follows a progressive deterioration of these artifacts due to design defects. These defects may have been introduced not only in the first design, but also during the maintenance of the software. The degradation of software design makes it even harder to understand the software and perform future maintenance, creating a vicious cycle. We propose a research plan to contribute in minimizing the degradation of software designs by providing an integrated system for the automatic detection and correction of design defects, along with monitoring the design quality. This system, called SUDERCO, is based on machine learning techniques and aims at providing a flexible and scalable tool to help reduce maintenance costs by preserving the design.

Keywords: design defect, SVM, recommender system, detection, correction, refactoring, Java.

TABLE DES MATIÈRES

RÉSUMÉ . . . . . **iii**

ABSTRACT . . . . . **iv**

TABLE DES MATIÈRES . . . . . **v**

LISTE DES TABLEAUX . . . . . **vii**

LISTE DES FIGURES . . . . . **viii**

CHAPITRE 1 : DÉFAUTS DE CONCEPTION ET QUALITÉ **1**

1.1 Contexte : maintenance des systèmes orientés objet 1

1.2 Problématique et objectifs 2

1.2.1 Détection des défauts à l'aide des SVM 2

1.2.2 Amélioration de la précision de la détection par la prise en compte des retours usagers 3

1.2.3 Correction des défauts à l'aide d'un système de recommandation basé sur l'AFC 3

1.2.4 Prise en compte de l'impact de la correction sur la qualité du système 5

1.2.5 Contribution du cadre SUDERCO 5

CHAPITRE 2 : ÉTAT DE L'ART **7**

2.1 Concepts préalables 7

2.1.1 Support Vector Machines (SVM) pour la classification 7

2.1.2 Analyse formelle de concepts pour la catégorisation 13

2.1.3 Systèmes de recommandation 14

2.2 État de l'art sur la détection des défauts de conception 16

2.3 État de l'art sur la correction des défauts de conception 18
2.4 Limites des travaux précédents 19

CHAPITRE 3 : SUDERCO : UN CADRE POUR LA DÉTECTION ET LA CORRECTION DES DÉFAUTS DE CONCEPTION . . . 21

3.1 SVMDetect : détection automatique des défauts au moyen des SVM avec prise en compte des retours usagers 22
3.1.1 Méthode 22
3.1.2 Expérimentation 24
3.2 FCAReCor : Système de recommandation pour la correction automatique basé sur l'AFC avec prise en compte des retours d'usagers 28
3.2.1 Méthode 28
3.2.2 Expérimentations 29
3.3 Mesure de l'impact des corrections grâce à PQMOD 31
3.3.1 Méthode 31
3.3.2 Expérimentation 31

CHAPITRE 4 : PLAN DE RECHERCHE 33

4.1 État actuel de notre travail de recherche 33
4.2 Plan de nos travaux futurs 34
4.3 Échéancier et plan de publications 35

BIBLIOGRAPHIE 37

LISTE DES TABLEAUX

2.I Description des animaux [18] 13

3.I Liste des defaults de conception [29] 26

3.II Liste des systèmes [29] 27

4.I Résultats de l'application de SVM pour la detection 34

4.II Plan de publications. 35

LISTE DES FIGURES

1.1 Exemple de diagramme de classe de blob 6

1.2 Correction du blob 6

2.1 Exemple d'un problème de discrimination à deux classes, avec un séparateur linéaire : la droite d'équation $y = x$. Le problème est linéairement séparable 9

2.2 L'hyperplan optimal avec la marge maximale et les vecteurs supports . 10

2.3 Changement de dimension pour résoudre le problème d'absence de séparateur linéaire 12

2.4 Treillis des animaux [18] 15

3.1 Étapes du cadre SUDERCO 22

4.1 Plan de recherche présent et futur 36

CHAPITRE 1

DÉFAUTS DE CONCEPTION ET QUALITÉ

1.1 Contexte : maintenance des systèmes orientés objet

Tout système orienté objet, depuis sa conception jusqu'à son retrait, passe par différents stades de son cycle de vie. Une des phases les plus importantes de ce cycle est la maintenance. En effet, la maintenance et la gestion de l'évolution du système après sa livraison constituent plus de 80% de l'ensemble des dépenses du cycle de développement [6, 14, 25]. La maintenance est donc l'activité la plus couteuse dans le cycle de vie d'un système. [37]

Les défauts de conception sont des mauvaises solutions à des problèmes récurrents de conception. Ils sont dans la plupart des cas la cause d'une faible maintenabilité des systèmes [24]. Au fur et à mesure que le système évolue, les efforts des mainteneurs se concentrent sur la correction des bogues au détriment des défauts de conception initiaux ou introduits au fil du temps. Le fait de se concentrer sur la correction des bogues au détriment de la correction des défauts de conception va entraîner la détérioration progressive de la structure du système [8] rendant ainsi sa compréhension et sa maintenance de plus en plus difficile. Ainsi, les mainteneurs ne peuvent plus garder une architecture cohérente et une conception adéquate surtout sous la pression des coûts et délais.

Il devient dès lors important, voir crucial de détecter ces défauts de conception et de les corriger dès le début du processus de développement du système. Aussi, la détection et la correction des défauts de conception au début du cycle de vie du système contribueront à réduire considérablement les coûts des activités de maintenance [37]. Pour améliorer la qualité du système et faciliter la tâche des mainteneurs dans la correction du système, il est important de détecter et de corriger les défauts le plus tôt possible dans le processus de développement comme le signale de façon très imagée et éloquente

Niccolo Machiavelli : “Certaines maladies, comme disent les médecins, sont faciles à soigner mais difficiles à reconnaitre à leur début, [...] mais au cours du temps quand elles n’ont pas été reconnues et traitées, elles deviennent faciles à reconnaitre mais difficiles à soigner”. [37].

1.2 Problématique et objectifs

1.2.1 Détection des défauts à l’aide des SVM

Pour faciliter la maintenance des systèmes plusieurs techniques ont été mises en place pour la détection des défauts de conception très tôt dans le processus de développent. Bon nombre de ces techniques se basent sur la définition manuelle et la mise en oeuvre de règles de détection des défauts de conception. Ces règles de détection sont figées et définies par l’utilisateur lui-même, nécessitant ainsi de lui une grande connaissance technique liée à la définition des défauts. Pourtant, les techniques d’apprentissage automatique se sont révélées adaptées à l’extraction automatique de connaissances. Aussi dans notre recherche abordons-nous les questions de recherche suivantes :

QR 1 : comment détecter les occurrences de défauts de conception dans les systèmes orientés objet sans avoir besoin de définir manuellement des règles pour la détection ? La detection des occurences des défauts de conception sur la base de leur description est clairement un problème de classification. À travers cette question de recherche, nous cherchons à montrer qu’il est possible de construire un classifieur, SVMDetect, basé sur les SVM (Support Vector Machines) pour la détection des défauts de conception. Ce classifieur n’aura pas besoin qu’on lui donne des règles pour la détection des défauts car il construira automatiquement une frontière de décision à partir d’une base d’apprentissage lui permettant par la suite de décider si une classe est un défaut ou pas.

QR 2 : quelle est l’efficacité de SVMDetect ? Pour montrer l’efficacité de SVMDetect,

nous calculerons la précision et le rappel que nous comparerons aux résultats des approches existantes dans la littérature. Nous proposerons une comparaison systématique avec tous les outils/approches.

1.2.2 Amélioration de la précision de la détection par la prise en compte des retours usagers

Dans l'extraction de connaissance par apprentissage supervisé, l'une des limitations est la petite taille de la base d'exemples car les utilisateurs ne sont pas assez patients pour étiqueter un grand nombre d'instances [26]. La conséquence de cette limitation est que le classifieur pourrait ne pas être bien entrainé et retourner donc un grand nombre de faux positifs. Les faux positifs sont les occurrences que l'algorithme de détection va considérer comme des défauts alors que ce ne sont pas des défauts. Ainsi une question dans notre recherche est la suivante :

QR 3 : comment utiliser la présence des faux positifs dans le résultat de la détection pour améliorer la qualité de la détection ? À travers cette question de recherche, nous cherchons à améliorer la précision et le rappel de SVMDetect dans la détection des défauts de conception en prenant en compte les retours des utilisateurs. En effet le fait de prendre en compte les évaluations des utilisateurs après la détection permettra d'agrandir la base d'apprentissage au fur et à mesure sans astreindre les utilisateurs à une tâche ardue de création d'une grande base d'apprentissage au début.

1.2.3 Correction des défauts à l'aide d'un système de recommandation basé sur l'AFC

Une fois les défauts détectés, il devient nécessaire de les corriger. Compte tenu de la taille des systèmes et du nombre de défauts présents dans les systèmes, il est crucial de proposer un outil qui permet de corriger automatiquement les défauts. Une correc-

tion manuelle ferait perdre beaucoup de temps et d'argent. La correction va consister à proposer une restructuration composée d'un ensemble de refactorisation. Les questions suivantes méritent d'être élucidées dans le cadre de ce volet de notre recherche :

QR 4 : comment exploiter les résultats de la détection dans la correction ? Nous exploitons les résultats fournis par SVMDetect pour caractériser les différentes classes de défauts et répertorier les attributs qui ont contribué le plus à leur détection. Moha *et al.* [30] l'ont fait pour le Blob. Nous allons donc étendre ce travail aux autres défauts de conception. La plupart des travaux ne font aucun lien entre la correction et la détection des défauts. Ces attributs serviront de critères pour juger de la pertinence des restructurations en évaluant si les valeurs de ces attributs se sont améliorées.

QR 5 : comment corriger automatiquement et efficacement les défauts détectés dans les systèmes ? Un aspect très important dans la correction est l'identification des restructurations à faire. Pour répondre à cette question nous proposons un système de recommandation, FCAReCor, qui utilisera l'Analyse Formelle de Concepts (AFC) et l'Analyse Relationnelle de Concepts (ARC) pour suggérer les restructurations afin de corriger les défauts et d'améliorer les caractéristiques détectées suite à l'application de SVMDetect.

QR 6 : comment exploiter les retours d'usagers pour améliorer la qualité des refactorisations ? Le système de recommande propose t-il les recommandation qui convient à l'utilisateur et la situation dans laquelle il se trouve ? Pour cela le système de recommandation FCAReCor va intégrer la prise en compte des retours d'usagers parce que la correction est subjective dans certains cas. Dans ces situations, il est important de pouvoir recueillir les préférences de l'utilisateur suite à une recommandation afin de raffiner les prochaines recommandations.

1.2.4 Prise en compte de l'impact de la correction sur la qualité du système

L'objectif de la correction du système est d'améliorer leur qualité. Ainsi, il est important de s'assurer que les refactorisations apportées améliorent effectivement la qualité des systèmes restructurés. Nous traiterons donc la question suivante :

QR 7 : Quel est l'impact des refactorisations sur la qualité des systèmes corrigés ? Pour traiter cette question nous utilisons le modèle de qualité développé par Khomh *et al.* [22], PQMOD dans leur méthode DEQUALITE, pour nous assurer que nous avons amélioré la qualité du système grâce aux corrections. Ainsi nous calculerons les facteurs de qualité avant et après la correction, nous en déduirons ensuite l'impact des refactorisations sur les systèmes corrigés. On pourra ensuite reprendre le processus jusqu'à atteindre une stabilité.

1.2.5 Contribution du cadre SUDERCO

Nous illustrons notre problématique et les contributions de notre approche, SUDERCO (« Système aUtomatique de Détection des défauts et de Recommandation de COrrections »), à travers cet exemple qui porte sur un défaut bien connu dans les systèmes orientés objet : Le blob. Le blob a été défini dans le tableau 3.I. Nous montrons comment notre approche permet de répondre aux questions de recherche. Dans la figure 1.1, nous avons un exemple de blob et la figure 1.2 est le résultat de sa correction à travers des opérations de refactorisations. Cet exemple tiré d'un système de gestion d'une bibliothèque est décrit dans [9].

Pour un tel exemple, notre approche consiste à extraire le blob en question grâce à SVMDetect. Ceci nous permettra de répondre au questions de recherche QR1, QR2. Ensuite l'utilisateur donne son avis sur le résultat de la détection. Cet avis est pris en compte pour améliorer les performances de SVMDetect [QR3]. À partir de cette détection, nous déduisons les caractéristiques importantes qui permettent cette détection. Les caractéristiques importantes pour le blob par exemple sont la cohésion mesurée par

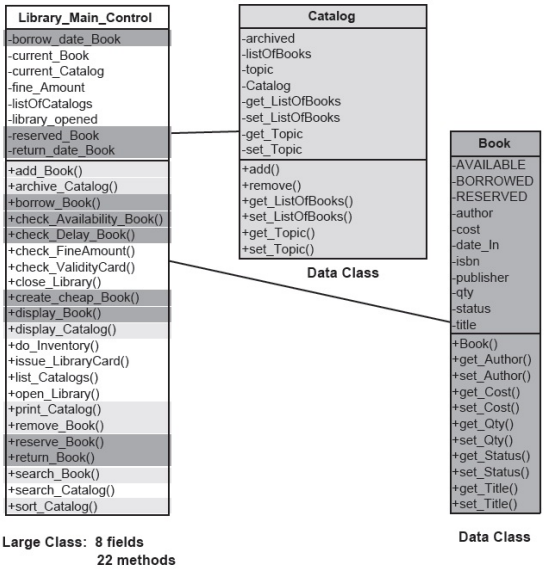

Figure 1.1 – Exemple de diagramme de classe de blob

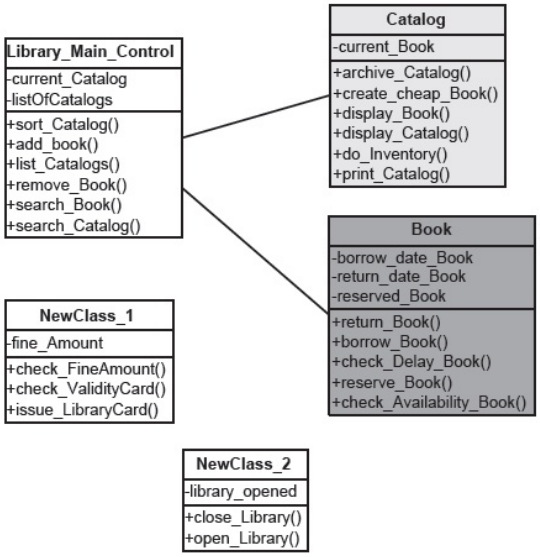

Figure 1.2 – Correction du blob

la métrique LCOM (Lack of COhesion Metric), et le couplage mesuré par la métrique CBO (Coupling Between Objects) [29] [Q4]. Ensuite, FCAReCor va se baser sur l'ARC pour identifier les restructurations qui améliorent le couplage et la cohésion et les recommander à l'usager [Q5, Q6] . Enfin, SUDERCO, grâce à PQMOD, va mesurer l'impact de cette correction sur la qualité du système [Q7].

CHAPITRE 2

ÉTAT DE L'ART

2.1 Concepts préalables

2.1.1 Support Vector Machines (SVM) pour la classification

Les algorithmes d'apprentissage automatique (machine-learning en anglais) ont pour objet d'extraire de la connaissance à partir de données. Les algorithmes d'apprentissage automatiques dit supervisés construisent un modèle de prédiction en utilisant un ensemble d'objets préalablement étiquetés, dit ensemble d'apprentissage. À partir de l'ensemble d'apprentissage, ou base d'exemples, l'algorithme estime les paramètres du modèle de prédiction les plus performants possible, c'est-à-dire qui produisent le moins d'erreurs en prédiction. À partir de ce modèle qui a été construit avec les paramètres optimaux, il sera possible de prédire la classe ou la catégorie d'un nouvel objet qui lui est soumis.

De façon formelle, on s'intéresse à un phénomène f (éventuellement non-déterministe) qui à partir d'une entrée x produit une sortie $y = f(x)$. Le but de l'algorithme est de déterminer une représentation compacte de f notée g et appelée fonction de prédiction, qui à une nouvelle entrée x associe une sortie $g(x)$. Pour déterminer la fonction g, on utilise un ensemble de couples entrée–sortie (x_n, y_n) (appelé base d'exemples) avec $x_n \in X$ et $y_n \in Y$. Le but de l'algorithme est donc de généraliser pour des entrées inconnues ce qu'il a pu « apprendre » grâce aux données de la base d'exemples, ceci de façon « raisonnable » en se basant sur des calculs de distance et de similarité. Un de ces algorithmes d'apprentissage automatique supervisé est Support Vector Machines (SVM) ou machines à vecteurs de support ou séparateurs à vaste marge. Les SVM sont un ensemble de techniques inspirées de la théorie statistique de l'apprentissage supervisé de Vladimir Vapnik introduite en 1995. Cet algorithme repose sur l'existence d'un classi-

fieur linéaire dans un espace approprié. Il fait appel à un jeu de données d'apprentissage pour apprendre les paramètres de ce classifieur. Il est également basé sur l'utilisation de fonctions dites noyau (kernel) qui permettent une séparation optimale des données.

Les SVM ont rapidement été adoptés pour leur capacité à travailler avec des données de grandes dimensions (un grand nombre d'attributs pour décrire chaque instance), le faible nombre de paramètres pour le modèle d'apprentissage, le fait qu'ils soient bien fondés théoriquement, et leurs bons résultats en pratique.

Les SVM ont été appliqués à de très nombreux domaines (bio-informatique, recherche d'information, vision par ordinateur, finance...). Selon les données, la performance des machines à vecteurs de support est de même ordre, ou même supérieure, à celle d'un réseau de neurones ou d'un modèle de mixture gaussienne [20]. Dans la présentation des principes de fonctionnement, nous schématiserons les données par des points dans un plan. Les SVM permettent de résoudre des problèmes de discrimination. Pour deux classes d'exemples donnés, le but des SVM est de trouver un classifieur linéaire, appelé hyperplan, qui va séparer les données en deux classes et maximiser la distance entre ces deux classes. Dans la figure 2.1, on détermine un hyperplan qui sépare les deux classes de points.

Pour un problème de discrimination à deux classes (discrimination binaire), on peut poser que $y \in \{-1, 1\}$, et le vecteur d'entrée x est dans un espace X muni d'un produit scalaire. On peut prendre par exemple $X = \mathbb{R}^N$.

Il est évident qu'il existe une multitude d'hyperplans valides mais la propriété remarquable des SVM est que l'hyperplan doit être optimal. La marge est la distance entre l'hyperplan et les échantillons les plus proches. Ces derniers sont appelés vecteurs de supports. On choisit donc l'hyperplan qui maximise la marge comme on le voit sur la figure 2.2. Ainsi, si un exemple n'est pas décrit parfaitement, une petite variation ne modifiera pas sa classification. L'hyperplan qui maximise la marge est donné par :

$$\arg\max_{w,w_0} \min_k \{||x - x_k|| : x \in R^N, w^T x + w_0 = 0\} \tag{2.1}$$

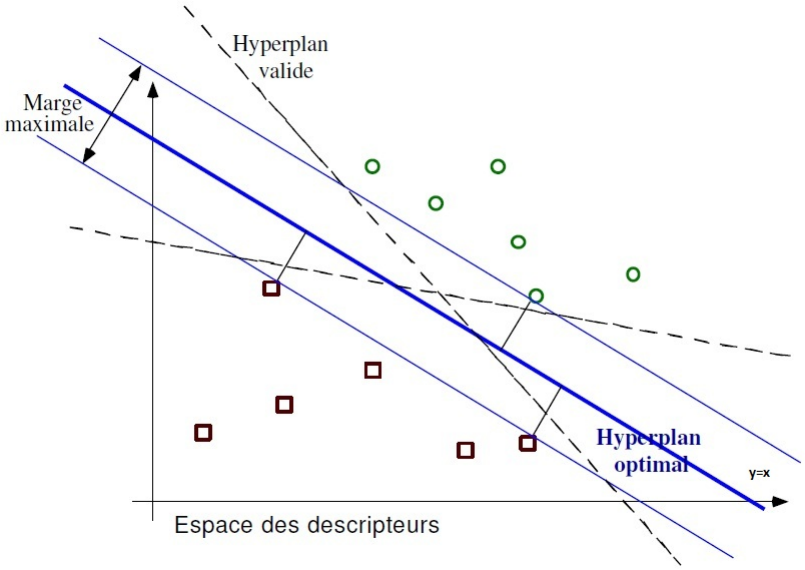

Figure 2.1 – Exemple d'un problème de discrimination à deux classes, avec un séparateur linéaire : la droite d'équation $y = x$. Le problème est linéairement séparable

Il s'agit donc de trouver w et w_0 remplissant ces conditions, afin de déterminer l'équation de l'hyperplan séparateur :

$$h(x) = w^T x + w_0 = 0 \tag{2.2}$$

La marge satisfait également la condition de séparabilité (à savoir $l_k(w^T x_k + w_0) \geq 0$). La distance d'un échantillon x_k à l'hyperplan est donnée par sa projection orthogonale sur l'hyperplan :

$$\frac{l_k(w^T x_k + w_0)}{||w||} \tag{2.3}$$

L'hyperplan séparateur (w,w_0) de marge maximale est donc donné par l'équation 2.4 :

$$\arg\max_{w,w_0} \left\{ \frac{1}{||w||} \min_k \left[l_k(w^T x_k + w_0) \right] \right\} \tag{2.4}$$

Afin de faciliter l'optimisation, on choisit de normaliser w et w_0, de telle manière que les échantillons à la marge (x^+_{marge} pour les vecteurs supports sur la frontière positive, et

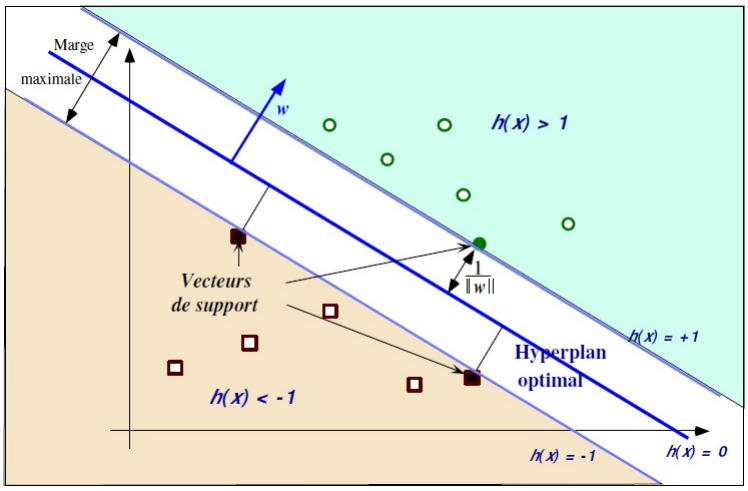

Figure 2.2 – L'hyperplan optimal avec la marge maximale et les vecteurs supports

x^-_{marge} pour ceux situés sur la frontière opposée) satisfassent :

$$\begin{cases} w^T x^+_{marge} + w_0 = 1 \\ w^T x^-_{marge} + w_0 = -1 \end{cases} \tag{2.5}$$

D'où pour tous les échantillons, k = 1, . . . , p

$l_k(w^T x_k + w_0) \geq 1$

Cette normalisation est la forme canonique de l'hyperplan ou hyperplan canonique. Avec la forme canonique, la marge vaut désormais $\frac{1}{||w||}$,. La recherche de l'hyperplan optimal revient donc à minimiser désormais $||w||$, soit à résoudre le problème d'optimisation dite formulation primale des SVM qui porte sur les paramètres w et w_0 (équation 2.6).

$$\begin{cases} \text{Minimiser} \quad \frac{1}{2}||w||^2 \\ \text{sous les contraintes} \quad l_k(w^T x_k + w_0) \geq 1 \end{cases} \tag{2.6}$$

Cette équation peut être résolue par la méthode classique des Multiplicateurs de Lagrange. Le lagrangien est la somme de la fonction objectif et d'une combinaison linéaire

des contraintes. Dans notre cas le lagrangien est donné par :

$$L(w, w_0, \alpha) = \frac{1}{2}||w||^2 - \sum_{k=1}^{p} \alpha_k \left\{ l_k(w^T x_k + w_0) - 1 \right\} \qquad (1) \qquad (2.7)$$

Pour trouver la solution du problème primal, il faut minimiser le lagrangien par rapport à w et w_0, et le maximiser par rapport aux α_i. Pour cette solution, les dérivées partielles du lagrangien s'annulent, selon les conditions de Kuhn-Tucker et on obtient :

$$\begin{cases} \sum_{k=1}^{p} \alpha_k l_k x_k & = w^* \\ \sum_{k=1}^{p} \alpha_k l_k & = 0 \end{cases} \qquad (2.8)$$

En réinjectant ces valeurs dans l'équation 2.7, on obtient la formulation duale du problème :

$$\begin{cases} \text{Maximiser} \quad \widetilde{L}(\alpha) = \sum_{k=1}^{p} \alpha_k - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j l_i l_j x_i^T x_j \\ \text{sous les contraintes} \quad \alpha_k \geq 0 \text{ , et } \sum_{k=1}^{p} \alpha_k l_k = 0 \end{cases} \qquad (2.9)$$

On résoud cette equation pour trouver les multiplicateurs de Lagrange optimaux α_k^* et w_0^*. On obtient ainsi l'hyperplan solution, en remplacçant w par sa valeur optimale w_0^*, dans l'équation de l'hyperplan $h(x)$, ce qui donne :

$$h(x) = \sum_{k=1}^{p} \alpha_k^* l_k (x \cdot x_k) + w_0^* \qquad (2.10)$$

Dans certains cas, comme montré sur la figure 2.3, nous ne sommes pas en mesure de trouver un hyperplan separateur linéaire. Afin de remédier au problème de l'absence de séparateur linéaire, l'idée des SVM est de reconsidérer le problème dans un espace de dimension supérieure, éventuellement de dimension infinie. Dans ce nouvel espace, on pourra trouver un séparateur linéaire.

On utilise une fonction noyau pour se ramener donc au cas d'un classifieur linéaire, sans changement d'espace. L'exemple le plus simple de fonction noyau est le noyau

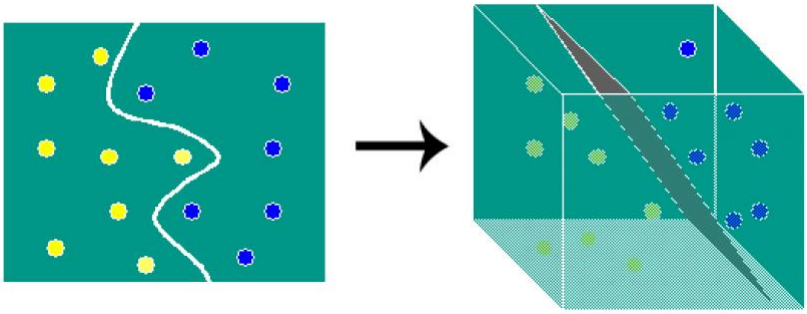

Figure 2.3 – Changement de dimension pour résoudre le problème d'absence de séparateur linéaire

linéaire utilisé dans le cas linéairement séparable : $K(x_i, x_j) = x_i^T * x_j$ Les noyaux usuels utilisés pour les SVM sont :

- le noyau polynomial : $K(x_i, x_j) = (x_i^T * x_j)^d$
- le noyau gaussien : $K(x, y) = exp(-\frac{||x-y||^2}{2\sigma^2})$

Dans le cas de l'utilisation d'une fonction noyau k, la solution est l'hyperplan séparateur d'équation :

$$h(x) = \sum_{k=1}^{p} \alpha_k^* l_k K(x, x_k) + w_0^* \tag{2.11}$$

Plusieurs méthodes ont été proposées également pour étendre les SVM au cas où plus de deux classes sont à séparer. Les deux plus connues sont appelées one versus all et one versus one. Formellement, les échantillons d'apprentissage et de test peuvent ici être classés dans M classes C1,C2,...,CM.

La méthode one-versus-all (appelée parfois one-versus-the-rest) consiste à construire M classifieurs binaires en attribuant le label 1 aux échantillons de l'une des classes et le label -1 à toutes les autres. En phase de test, le classifieur donnant la valeur de confiance (e.g la marge) la plus élevée remporte le vote.

La méthode one-versus-one consiste à construire $\frac{M*(M-1)}{2}$ classifieurs binaires en confrontant chacune des M classes. En phase de test, l'échantillon à classer est analysé par chaque classifieur et un vote majoritaire permet de déterminer sa classe.

2.1.2 Analyse formelle de concepts pour la catégorisation

L'AFC est une méthode qui utilise l'apprentissage non supervisé pour identifier des catégories d'objets ayant des propriétés communes. L'AFC prend en entrée un tableau de dimension deux. Ce tableau est appelé contexte formel binaire et définit une relation d'incidence entre les objets et les propriétés de ces objets. La table 2.I présente un exemple de contexte formel.

Définition 2.1.1 (Contexte Formel)**.** *Un contexte formel est un triplet $K = (O, A, R)$, où :*

- *O est l'ensemble des objets ;*
- *A est l'ensemble des attributs (caractéristiques) ;*
- *$R \subseteq O \times A$ est une relation tel que $\forall (o, a) \in R$, a est un attribut de l'objet o.*

Tableau 2.I – Description des animaux [18]

	vole (v)	nocturne (n)	plume (p)	migrateur (m)	bec-plat (b)	membrane (me)
palatouche (P)	x					x
chauve-souris (C)	x	x				
autruche (A)			x			
flamant-rose (F)	x		x	x		
goéland (G)	x		x		x	

Un concept permet de faire un regroupement maximal des objets partageant le même ensemble de propriétés. Par exemple : *flamant-rose, goeland* partagent les propriétés $vole, plume$ car aucun autre animal ne possède ces deux attributs et ces deux animaux n'ont rien de plus en commun. De façon formelle, on note f et g deux applications caractéristiques de R :

$f : \mathscr{P}(O) \rightarrow \mathscr{P}(A)$

$X \longmapsto \{y \in A \mid \forall x \in X, (x, y) \in \mathscr{R}\}$

$g : \mathscr{P}(A) \rightarrow \mathscr{P}(O)$
$Y \longmapsto \{x \in O \mid \forall y \in Y, (x,y) \in \mathscr{R}\}$

Définition 2.1.2 (Concept)**.** *Un concept est une paire $C = (X,Y)$ avec $X \subseteq O$, $Y \subseteq A$ et $X = \{o \in O | \forall y \in Y, (o,y) \in I\}$ est l'extension (objets couverts), notée $Ext(C)$ $Y = \{a \in A | \forall x \in X, (x,a) \in I\}$ est l'intension (attributs partagés), notée $Int(C)$.*

On a la propriété, pour tout concept $C = (X,Y)$, $Y = f(X)$ et $X = g(Y)$.

Définition 2.1.3 (Treillis de concepts, treillis de Galois d'une relation binaire)**.** *Le treillis de concepts $\mathscr{L}$ est l'ensemble des concepts formels de $K = (O,A,R)$, ordonné par la relation d'ordre partiel : c_1 spécialise c_2 ($c_1 \leq_{\mathscr{L}} c_2$) si l'extension de c_1 est incluse dans celle de c_2 (et inversement, l'intension de c_2 est incluse dans celle de c_1). $\mathscr{L}$ possède au plus $2^{min(|O|,|A|}$ concepts, car il se plonge dans le treillis des parties de O et dans le treillis des parties de A.*

La figure 2.4 correspond au contexte formel du tableau 2.I

2.1.3 Systèmes de recommandation

Les systèmes de recommandation permettent le filtrage de l'information pour proposer à l'utilisateur des informations qui seront pertinentes. Le système de recommandation pour faire la recommandation de ressources se base sur le profil de l'utilisateur, les caractéristiques des objets à recommander et l'environnement de l'utilisateur. De ces points de vue, les systèmes de recommandation sont généralement regroupés en trois catégories :

- La recommandation basée sur le contenu qui va utiliser les caractéristiques des objets à recommander. Dans cette catégorie, on recommande à l'utilisateur les objets qui sont similaires aux objets qu'il a déjà préférés dans le passé [4].

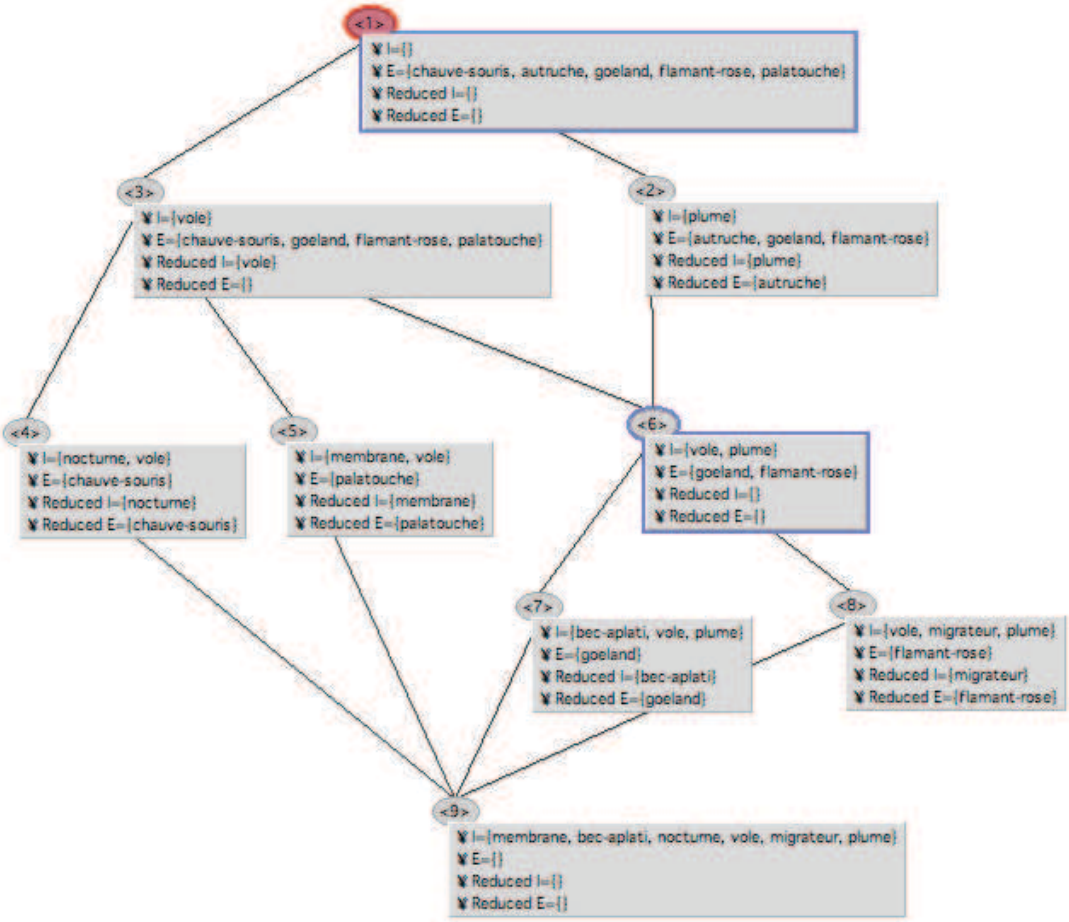

Figure 2.4 – Treillis des animaux [18]

- Le filtrage collaboratif basé sur le profil de l'utilisateur. On va recommander à l'utilisateur les objets qui ont été appréciés par les utilisateurs dont les profils sont similaires à son profil [34].

- Les approches hybrides. Ces approches combinent la recommandation basée sur le contenu et le filtrage collaboratif [4].

Pour plus de details sur les systèmes de recommandation de façon générale on peut se referer aux articles [3] , [2], [1].

2.2 État de l'art sur la détection des défauts de conception

De nombreuses approches ont été proposées dans la littérature pour la détection des défauts de conception.

Travassos *et al.* [44] ont créé un ensemble de techniques de lecture pour aider à l'inspection manuelle afin de détecter les défauts dans les conceptions orientés objets. Cette approche reste manuelle et est difficilement applicable à des systèmes de grandes tailles.

Marinescu [27] quant à lui a défini une stratégie de détection pour la formulation de règles basées sur les métriques pour détecter les défauts. Dans cette stratégie, il utilise un mécanisme de filtrage permettant de réduire la taille de l'ensemble de données puis un mécanisme de composition de métriques grâce à des operateurs ensemblistes. Cette approche même si elle a été implantée et automatisée n'a été validée que sur un seul système. Aussi, comme l'a souligné Moha [29], il n'existe pas de plateforme de détection sous jacente à cette stratégie ainsi que de processus faisant le lien entre les spécifications des défauts et leur détection.

Munro [33] a défini une approche similaire à la stratégie de détection de Marinescu. Il établit une description assez précise des défauts en suivant un patron pour la description. Ensuite, il propose des heuristiques basées sur les valeurs des métriques permettant de

détecter les défauts. Cependant, il n'existe pas une automatisation de cette technique pour la détection des défauts.

AliKacem *et al.* [5] proposent une approche basée sur les règles qui utilisent les métriques, les relations d'héritage et la logique flou pour détecter les violations des principes de qualité. Cette approche n'a toutefois pas encore été validée.

Dhambri *et al.* [11] proposent, pour la détection des défauts de conception, une technique basée sur la visualisation. Cette technique qui est une approche semi automatique nécessite toutefois l'intervention d'un expert humain pour la détection visuelle des anomalies.

Moha *et al.* [29–31] ont proposé la technique DETEX pour la détection de défauts dans le cadre de leur méthode DECOR pour la détection et la correction des défauts dans les systèmes orientés objet. Ce travail sur lequel nous basons nos travaux est le plus élaboré dans ce domaine. DETEX est une approche basée sur la génération automatique d'algorithmes de détection issus de fiches de règles. DETEX est appliqué en trois étapes : à partir de l'analyse du domaine, on définit les règles qui permettront de détecter les occurrences des défauts dans les systèmes. Ensuite, à partir des ces règles, on génère automatiquement les algorithmes de détection qui sont ensuite appliqués sur des modèles des systèmes afin d'obtenir les occurrences des défauts présents.

Cette approche bien qu'ayant automatisé beaucoup d'étapes reste quand même difficile à mettre en oeuvre de façon complètement automatique. En effet, l'analyse du domaine nécessite beaucoup d'efforts de recherche dans la littérature pour bien caractériser les défauts. La classification et la définition des défauts se fait manuellement. Une grande limitation de cette approche est que les résultats et les règles de la détection ne sont pas mis à profit dans la correction ; même si les deux ont été inclus dans la méthode, ils demeurent deux processus séparés.

2.3 État de l'art sur la correction des défauts de conception

Dans la littérature nous trouvons peu de travaux qui se sont intéressés à l'automatisation des corrections des défauts de conception. La correction est une activité qui se divise en trois grandes étapes importantes [28] : (1) déterminer les modifications à faire pour corriger le défaut ; (2) appliquer les refactorisations sur le système ; (3) garantir que les refactorisations preserve le comportment du système.

L'étape la plus complexe et la moins automatisée est l'étape (1). L'étape (2) se résume en l'application des refactorisations qui ont été très bien étudiées et détaillées par les travaux de Opdyke [35] et étendues par Fowler dans son livre [15]. Une refactorisation est une modification du code source d'un programme afin d'améliorer sa structure sans changer son comportement externe [35]

Dans le cadre de l'étape (1), Trifu *et al.* [45] ont proposé pour chaque défaut un exemple d'implantation du système afin d'éviter ce défaut. Cependant, ils ne proposent pas une stratégie permettant au développeur de corriger automatiquement le défaut particulier présent dans son système.

Sahraoui *et al.* [41] combinent le calcul de métriques, l'analyse formelle de concepts et d'autres techniques de graphe dans leur approche pour identifier des objets dans du code procédural. Cette approche qui utilise du code procedural ne peut pas s'appliuqer en l'état à la detection des défauts de conception qui s'applique aux systèmes orientés objets.

Godin et Mili [16] utilisent l'analyse formelle de concepts avec les treillis de gallois pour trouver des restructurations de hiérarchies utiles et redistribuer les membres. Cependant, cette approche ignore les relations entre les éléments des classes. Une variante de cette approche a été utilisée par la suite par Snelting et Tip [42]. Elle est basée également sur l'AFC.

Kirk *et al.* [23] proposent une approche qui regroupe les attributs d'une classe en sous ensemble. Cette approche a été conçue pour traiter les larges classes. C'est une ap-

proche intéressante mais qui n'est pas applicable directement à notre problème puisque nous nous intéressons à la correction des défauts de conception pouvant inclure plusieurs classes.

Les travaux qui se rapprochent le plus de nos travaux et dont nous nous sommes d'ailleurs inspirés sont les travaux de Moha *et al.* [30]. Ainsi Moha *et al.* [30] proposent CorEx qui utilise l'ARC (une extension de l'AFC) pour corriger les défauts. Cette technique de correction s'articule autour de quatre étapes : (1) identification des entités liées au défaut, (2) extraction de familles de contextes relationnels qui utilisent les membres de classes, (3) dérivation du trellis, (4) exploration du trellis pour obtenir un ensemble de restructurations. Le problème est que cette approche n'a été mise en place que pour le blob. Aussi, aucun lien n'est fait entre les résultats de la détection et la correction. Enfin, le parcours du treillis et l'application des refactorisations restent manuelles.

Somme toute, tous les travaux faits dans ce domaine suggèrent que l'AFC est le cadre idéal pour s'attaquer aux problèmes de correction.

2.4 Limites des travaux précédents

Nous mentionnons dans cette section les limites des approches proposées dans la littérature et comment, dans SUDERCO, nous comptons résoudre ces limitations.

1. Les règles de détection telle que définies par Moha *et al.* [29–31] sont figées et issues d'une analyse fastidieuse du domaine. C'est pourquoi dans SUDERCO, nous proposons une technique de détection basée sur les SVM, qui sont un algorithme d'apprentissage automatisé.

2. Non possibilité de l'évolution des règles au cours de l'exécution du programme de détection. Pour toutes les approches proposées, nous ne sommes pas en mesure de modifier automatiquement les règles de détection si nous nous rendons compte qu'il y a beaucoup de faux positifs. Pour remédier à ce problème, nous proposons la prise en compte des retours usagers dans SVMDetect. Ceci aura pour avantage

d'agrandir la base d'exemples de l'algorithme et lui permettra ainsi de redéfinir automatiquement la frontière de décision pour corriger les faux positifs.

3. Dans la plupart des approches dans la littérature, la correction des défauts reste encore manuelle, totalement ou en grande partie. Ceux qui ont essayé d'automatiser la correction ne l'ont fait que pour un seul défaut. C'est le cas du blob dans [29, 30]. Nous proposons une étude complète de quatre défauts de conception bien connus afin d'en dégager leurs caractéristiques et proposer un système de recommandation des corrections, FCAReCor, qui prenne en compte tous ces défauts et propose automatiquement des refactorisations pour la correction.

4. Non prise en compte du retour des usagers pour améliorer la correction. Nous prévoyons dans FCAReCor de prendre en compte les retours usagers pour agrandir la base d'exemples et améliorer la recommandation.

5. Pas de lien entre la détection des défauts et leur correction dans les approches proposées. SUDERCO se veut un cadre intègrant détection et correction. Ainsi, grâce à l'analyse des résultats de SVMDetect, nous sommes en mesure de déterminer les facteurs importants dans la classification de chaque défaut. Ces facteurs déterminant seront des éléments important pour le choix d'une restructuration donnée.

6. Pas de lien entre la qualité du système et la correction apportée. Notre cadre SUDERCO va intégrer un modèle de qualité permettant ainsi de mesurer les facteurs de qualité du système avant la correction et après la correction pour s'assurer que la correction ne dégrade pas le niveau de qualité du système.

CHAPITRE 3

SUDERCO : UN CADRE POUR LA DÉTECTION ET LA CORRECTION DES DÉFAUTS DE CONCEPTION

Afin de réduire les coûts liés à la gestion de l'évolution des systèmes et à leur correction, nous proposons de mettre en place un cadre intégré, SUDERCO (« Système aUtomatique de Détection des défauts et de Recommandation de COrrections »), pour la détection et la correction des défauts de conception. SUDERCO intègrera (1) la détection automatique des défauts de conception à l'aide de l'algorithme d'apprentissage supervisé SVM, (2) leur correction automatique à l'aide d'un système de recommandation basé sur l'AFC, (3) la mesure de l'impact des corrections sur la qualité des systèmes et(4) la prise en compte des retours d'usagers à chaque étape pour améliorer les performances du système.

Dans ce chapitre, nous présentons d'abord notre démarche méthodologique générale pour la mise en place de SUDERCO, puis nous abordons les étapes méthodologiques pour la mise en œuvre de chacune des composantes de SUDERCO.

Notre démarche méthodologique consiste en trois étapes répondant chacune à un point important de notre problématique :

1. La détection des défauts de conception au moyen d'un algorithme d'apprentissage automatique : les SVM. Notre algorithme, SVMDetect, est moins rigide que les fiches de règles de DECOR par exemple. En effet, nous n'avons pas besoin de définir manuellement et continuellement les règles de détection qui peuvent évoluer. Grâce à l'algorithme d'apprentissage supervisé, la frontière de décision sera améliorée au fur et à mesure que la base d'apprentissage va grossir. Aussi, un aspect nouveau est que nous comptons améliorer la précision de la détection en prenant en compte le retour d'usager.

2. La correction automatique des défauts détectés à travers un système de recommandation de restructurations. Notre système de recommandation, FCAReCor, s'appuie sur l'Analyse Formelle des Concepts (AFC). Aussi, FCAReCor va être amélioré continuellement en prenant en compte les retours d'usagers.

3. L'évaluation de l'impact des corrections sur la qualité des systèmes. En utilisant le modèle de qualité, PQMOD, proposé par par Khomh *et al.* [22], nous analyserons si la qualité du système a été ameliorée suite à la correction.

Nous présentons dans la figure 3.1 les étapes du cadre SUDERCO.

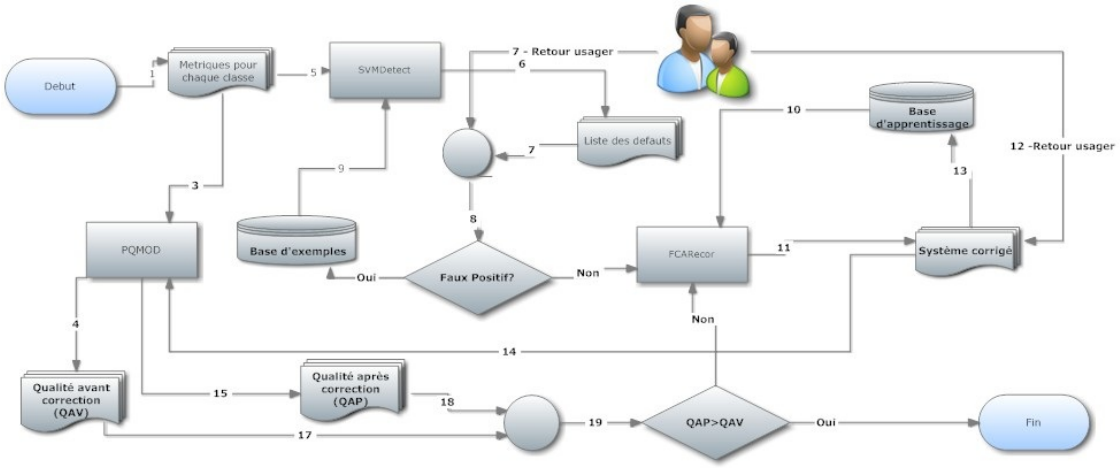

Figure 3.1 – Étapes du cadre SUDERCO

Dans ce qui suit, les principales étapes de notre démarche vont être étayées.

3.1 SVMDetect : détection automatique des défauts au moyen des SVM avec prise en compte des retours usagers

3.1.1 Méthode

L'approche de détection des défauts de conception de SUDERCO, SVMDetect, utilise l'algorithme d'apprentissage supervisé SVM (Support Vector Machines). Soit DB-Defect, une base d'exemples formée d'un ensemble de classes issues d'un système

orienté objet. Pour chaque classe C_i de DBDefect, nous calculons un ensemble de métriques orientées objet. Chaque classe est aussi évaluée comme appartenant à une catégorie de défauts de conception (Blob, FunctionalDecomposition, Spaguetticode, SwissArmyKnife et Sain). La catégorie Sain permet de caractériser les classes qui ne sont pas des défauts. Soit DBTest, l'ensemble des classes d'un système sur lequel on souhaite appliquer SVMDetect pour détecter les défauts de conception. Pour chacune de ces classes, nous calculons les mêmes métriques que pour la base DBDetect.

SVMDetect se divise en six étapes. Les étapes de (1) spécification des métriques et de (2) définition des catégories qui peuvent se faire simultanément permettent de construire les bases DBDefect et DBTest. Ensuite, l'étape (3) est dédiée à la construction du classifieur SVM, suivie (4) de l'évaluation du modèle, (5) de la prise en compte des retours usagers et enfin (6) de l'évaluation de l'impact de la prise en compte des retours usagers. Ces différentes étapes sont décrites dans ce qui suit.

Étape 1 (Spécification des métriques) : Il faut définir les entrées de SVMDetect. SVMDetect prend en entrée la base d'exemples DBDectect et la base de test DBTest. Pour construire la base d'apprentissage DBDetect, nous calculons un ensemble de métriques orientées objet qui serviront d'attributs x_n pour les classes. Ces attributs permettront de caractériser les défauts à détecter. Pour chaque défaut de conception, on définit un ensemble de métriques pertinentes à prendre en compte pour la détection. Nous construisons ensuite DBTest en calculant l'ensemble de ces métriques pour le système sur lequel SVMDetect doit être appliqué.

Étape 2 (Définition des catégories) : l'algorithme d'apprentissage automatique pour faire la classification a besoin de connaitre les différentes catégories existantes dans la base d'exemples. À cette étape, nous définissons quatre catégories qui sont les classes de défauts de conception les mieux connus qui devront être détectés par l'algorithme de détection SVMDetect. Ensuite, il faut taguer chaque classe comme étant un défaut ou pas afin de compléter la base d'apprentissage DBDetect.

Étape 3 (Construction du classifieur SVM) : nous construisons le classifieur SVM-Detect sur la base d'exemples DBDetect qui a été construite à l'étape 1 et 2. À cette étape, nous construisons deux versions de SVMDetect : SVMDetect Binaire qui un classifieur SVM binaire qui permettra de détecter un seul défaut à la fois et SVMDetect multiClasse, classifieur SVM Multi classses, afin de détecter tous les défauts en même temps. Ensuite nous appliquons ce classifieur sur les bases DBTest construites à l'étape 1.

Étape 4 (Évaluation du modèle) : évaluer les performances du modèle en calculant la précision et le rappel de la détection.

Étape 5 (Prise en compte des retours d'usagers) : les résultats de la détection sont proposés aux usagers pour évaluation. Dans ce cas, les usagers pourront identifier les faux positifs. Ces retours des usagers seront utilisés pour enrichir la base d'exemples DBDetect en lui ajoutant ces nouvelles classes taguées et permettre ainsi à SVMDetect de redéfinir la frontière de décision avec plus de précision.

Étape 6 - (Évaluation de l'impact de la prise en compte des retours d'usagers) : Après avoir appliqué SVMDetect sur la base DBTest1, on enrichit la base d'exemples DBDetect avec les évaluations de l'usager, puis on mesure les performances de SVMDetect sur DBTest2 après avoir appris sur la base DBDetect enrichie par les évaluations des usagers.

3.1.2 Expérimentation

SVMDetect sera validé en l'appliquant sur dix systèmes Java libres pour la détection de quatre défauts de conception afin de comparer nos résultats à DECOR. Pour cette validation, nous allons mener une expérimentation. Dans ce qui suit, nous définissons les différents éléments qui participent à la réalisation de cette expérimentation.

3.1.2.1 Questions de recherche et hypothèses des expérimentations

Ces expérimentations nous permettront de répondre à nos questions de recherche et par la suite de valider nos hypothèses d'étude.

QR 1 : comment détecter les occurrences de défauts de conception dans les systèmes orientés objet sans avoir besoin de définir manuellement des règles pour la détection ? La détection des occurrences des défauts de conception sur la base de leur description est clairement un problème de classification. À travers cette question de recherche nous cherchons donc à montrer qu'il est possible de construire un classifieur, SVMDetect, basé sur les SVM pour la détection des défauts de conception. Ce classifieur n'aura pas besoin qu'on lui donne des règles pour la détection des défauts car il construira automatiquement une frontière de décision à partir d'une base d'apprentissage lui permettant par la suite de décider si une classe est un défaut ou pas.

QR 2 : quelle est l'efficacité de SVMDetect ? Pour montrer l'efficacité de SVMDetect, nous calculerons la précision et le rappel que nous comparerons aux résultats des approches existantes dans la littérature. Nous proposerons une comparaison systématique avec les tous les outils/approches.

QR 3 : comment utiliser la présence des faux positifs dans le résultat de la détection pour améliorer la qualité de la détection ? À travers cette question de recherche, nous cherchons à améliorer la précision et le rappel de SVMDetect dans la détection des défauts de conception en prenant en compte les retours des utilisateurs. En effet, le fait de prendre en compte les évaluations des utilisateurs après la détection permettra d'agrandir la base d'apprentissage au fur et à mesure sans astreindre les utilisateurs à une tâche ardue de création d'une grande base d'apprentissage au début.

3.1.2.2 Sujets des expérimentations

SVMDetect sera utilisé dans cette expérimentation pour la détection de quatre défauts bien connus, Le Blob, Le Functional Decomposition, Le Spaghetti Code et Le Swiss Army Knife. Une brève description de ces défauts est donnée dans le tableau 3.I

Le *Blob* (également appelé God class [39]) correspond à une large classe contrôleur qui dépend de données localisées dans des classes de données associées. Une large classe déclare beaucoup d'attributs et de méthodes et a une faible cohésion. Une classe contrôleur monopolise la plupart du traitement réalisé par le système, prend la plupart des décisions et dirige de près le traitement des autres classes. [47] Les classes contrôleurs sont identifiables par des noms suspects tels que `Process`, `Control`, `Manage`, `System`, etc. Une classe de données contient seulement des données et réalise peu de traitement sur ces données. Elle est composée d'attributs et de méthodes accesseurs fortement cohésifs.
Le *Functional Decomposition* peut se produire si des développeurs expérimentés en procédural avec une petite connaissance de l'orienté objet implémentent un système orienté objet. Brown décrit cet anti-patron comme une routine principale qui appelle de nombreuses sous-routines. Le Functional Decomposition correspond à une classe principale avec un nom procédural, tel que `Compute` ou `Display`, dans lequel l'héritage et le polymorphisme sont à peine utilisés. Cette classe est associée à de petites classes, qui déclarent beaucoup d'attributs et implémentent peu de méthodes.
Le *Spaghetti Code* est un anti-patron qui est caractéristique d'une pensée procédurale dans la programmation orientée objet. Le Spaghetti Code se révèle par des classes sans structure, déclarant de longues méthodes avec pas de paramètres et utilisant des variables de classes. Les noms des classes et méthodes peuvent suggérer de la programmation procédurale. Le Spaghetti Code n'exploite pas et empêche l'utilisation de mécanismes orientés objet tels que le polymorphisme et l'héritage.
Le *Swiss army knife* fait référence à un outil répondant à un large éventail de besoins. L'anti-patron Swiss Army Knife correspond à une classe complexe qui offre un grand nombre de services, par exemple, une classe complexe implémentant un grand nombre d'interfaces. Le Swiss Army Knife est différent du Blob car il expose une grande complexité pour adresser tous les besoins prévisibles d'une partie d'un système, alors que le Blob est un singleton monopolisant tout le traitement et les données d'un système. Ainsi, plusieurs Swiss Army Knives peuvent exister dans un système, par exemple, les classes 'utilitaires'.

Tableau 3.I – Liste des defaults de conception [29]

3.1.2.3 Objets et processus des expérimentations

Pour des fins de comparaison de nos résultats aux résultats de DECOR, les objets des expérimentations seront dix systèmes libres Java : ArgoUML, Azureus, GanttProject, Log4J, Lucene, Nutch, PMD, QuickUML et deux versions de Xerces. Ces systèmes sont décrits dans le tableau 3.II

Pour chacun de ces 10 systèmes une série de plus de cinquante métriques est calculée à l'aide de POM. Les principales sont LCOM et LOC. Ce sont ces métriques qui vont servir à construire les bases DBDetect et DBTest. Pour tous ces systèmes, des ingénieurs logiciels indépendants ont validé les quatre défauts (le blob, le Functional Decomposition, le Spaghetti Code et le Swiss Army Knife) que nous cherchons à détecter. Ainsi,

Nom	Version	Lignes de code	Nombre de classes	Nombre d'interfaces
ArgoUML	0.19.8	113,017	1,230	67
Un outil de modélisation UML				
Azureus	2.3.0.6	191,963	1,449	546
Un client pair- à-pair implémentant le protocole BitTorrent				
GanttProject	1.10.2	21,267	188	41
Un outil de gestion de projet pour réaliser des diagrammes de GANTT				
Log4J	1.2.1	10,224	189	14
Un framework extensible pour logger et débugger les applications Java				
Lucene	1.4	10,614	154	14
Un moteur de recherche textuelle en JAVA				
Nutch	0.7.1	19,123	207	40
Un moteur de recherche Web bas´e sur LUCENE				
PMD	1.8	41,554	423	23
Un outil d'analyse de code source et détection de bugs				
QuickUML	2001	9,210	142	13
Un outil de modélisation UML pour les diagrammes de classes et de séquences				
Xerces	1.0.1	27,903	189	107
Un plate-forme pour construire des analyseurs syntaxiques XML en JAVA				
Xerces	2.7.0	71,217	513	162
Version 2.7.0 de l'analyseur syntaxique XERCES				

Tableau 3.II – Liste des systèmes [29]

nous pouvons calculer sur l'ensemble de ces systèmes les mesures de précision et de rappel pour valider notre approche. Pour chaque système et pour la détection de chaque défaut, nous calculons la précision et le rappel avant la prise en compte des retours d'usagers et après la prise en compte des retours d'usagers.

$$Precision = \frac{NVP}{NVP + NFP} \tag{3.1}$$

$$Rappel = \frac{NVP}{NVP + NFN} \tag{3.2}$$

Avec :

- NVP : Nombre de Vrai Positifs, les occurrences qui sont détectées comme des défauts alors qu'elles sont réellement des défauts ;
- NFP : Nombre de Faux Positifs, les occurrences qui sont détectées comme des défauts alors qu'elles ne le sont pas réellement ;
- NFN : Nombre de Faux Négatifs, les occurrences qui ne sont pas détectées alors qu'elles sont en réalité des défauts.

3.2 FCAReCor : Système de recommandation pour la correction automatique basé sur l'AFC avec prise en compte des retours d'usagers

3.2.1 Méthode

Selon [28], tout processus de refactorisations doit suivre les étapes suivantes :

1. Identifier les parties du système qui nécessite une refactorisation.
2. Garantir que l'application de ces refactorisations préserve le comportement du système.
3. Appliquer les refactorisations.
4. Évaluer l'effet des refactorisations sur les caractéristiques de qualité du système.
5. Maintenir la cohérence entre le code restructuré et les autres artefacts du système (comme la documentation, les documents de conception, tests, etc.).

Nous proposons un système de recommandation de corrections, FCAReCor, dans le cadre de SUDERCO, qui automatise les étapes 1 à 4 du processus de refactorisations. La mise en place de FCAReCor se fera selon les étapes suivantes :

Étape 1 (Définition des caractéristiques des objets) : à partir des caractéristiques à améliorer propres à chaque défaut, définir les caractéristiques des objets qui serviront à définir la recommandation. Dans notre cas, les objets à recommander sont les restructurations.

Étape 2 (Caractérisation des défauts) : construire une base d'exemples avec pour chaque défaut un ensemble de restructurations possibles en fonction des caractéristiques du système et du défaut considéré.

Étape 3 (Construction du système de recommandation) : construire un système de recommandation basé sur le contenu qui utilise les caractéristiques définies dans les premières étapes.

Étape 4 (Identification des parties du système à restructurer) : dans cette étape, nous identifions les parties du système concernées par la refactorisation. Pour cela, nous utiliserons une combinaison de plusieurs méthodes en fonction des caractéristiques du défaut à corriger. Par exemple, nous utiliserons une version améliorée de l'ARC proposée par [29] en prenant en compte les informations liées à l'accès des attributs et l'invocation des méthodes par d'autres méthodes dans les classes associées.

Étape 5 (Choix des restructurations à proposer) : proposer des restructurations à recommander à partir de la base qui a été construite aux étapes précedentes.

Étape 6 (Mesure de l'impact de la correction sur la qualité) : nous presumons que cette mesure pourrait se faire par des techniques d'analyse dynamique de code (que nous devrons identifier) pour nous assurer que le comportement du système est préservé par les restructurations.

3.2.2 Expérimentations

3.2.2.1 Questions de recherche et hypothèses des expérimentations

Les experimentations que nous menerons dans le cadre de la correction nous permettront de répondre aux questions de recherche suivantes :

QR 4 : comment exploiter les résultats de la détection dans la correction ? Nous exploitons les résultats fournis par SVMDetect pour caractériser les différentes classes de défauts et répertorier les attributs qui ont contribué le plus à leur détection. Moha *et al.* [30] l'ont fait pour le Blob. Nous allons donc étendre ce travail aux autres défauts de conception. La plupart des travaux ne font aucun lien entre la correction et la détection des défauts. Ces attributs serviront de critères pour juger de la pertinence des restructurations en evaluant si les valeurs de ces attributs se sont ameliorées.

QR 5 : comment corriger automatiquement et efficacement les défauts détectés dans les systèmes ? Un aspect très important dans la correction est l'identification des restructurations à faire. Pour répondre à cette question nous proposons un système de recommandation, FCAReCor, qui utilisera l'Analyse Formelle de Concepts et l'Analyse Relationnelle de Concepts pour suggérer les restructurations afin de corriger les défauts et d'améliorer les caractéristiques détectées suite à l'application de SVMDetect.

QR 6 : comment exploiter les retours d'usagers pour améliorer la qualité des refactorisations ? Le système de recommandation propose t-il les recommandations qui conviennent à l'utilisateur et à la situation dans laquelle il se trouve ? Pour pouvoir staisfaire aux exigences, le système de recommandation FCAReCor va intégrer la prise en compte des retours d'usagers parce que la correction est subjective dans certains cas. Dans ces situations, il est important de pouvoir recueillir les préférences de l'utilisateur suite à une recommandation afin de raffiner les prochaines recommandations.

3.2.2.2 Sujets des expérimentations

Nous utilisons ici les mêmes sujets (les quatre défauts) extraits par SVMDetect pour lesquels nous proposerons des corrections.

3.2.2.3 Objets et processus des expérimentations

Nous utilisons ici également les 10 systèmes libres Java utilisés précédemment. Nous mettrons en œuvre notre système, FCAReCor, avec un échantillon de d'usagers sur des systèmes sur lesquelles nous aurons aura au préalable détecté des défauts. Le système proposera des corrections pour ces défauts et l'usager évaluera la proposition du système de recommandation. Ceci nous permettra d'améliorer le système de recommandation et d'en mesurer les performances.

3.3 Mesure de l'impact des corrections grâce à PQMOD

3.3.1 Méthode

Dans cette section, nous entendons par qualité, la mesure de la prédisposition d'une classe aux fautes et aux changements. Pour évaluer cette mesure et comprendre l'impact des corrections sur les systèmes, nous utiliserons la modèle de qualité PQMOD proposée par Khomh [21]. La plupart des modèles de qualité utilisent uniquement des métriques de classes ou de relations pour mesurer la qualité des systèmes [21] c'est pourquoi nous avons choisi en particulier PQMOD car il intègre dans l'évaluation de la qualité, en plus des métriques, la façon dont les classes sont organisées ; organisation impactée par les restructurations.

Nous présumons que nous pouvons appliquer ce modèle avant la correction et après la correction pour voir l'evolution des attributs de qualité.

Dans nos travaux futurs, nous explorerons les moyens d'exploiter cette évolution pour au besoin modifier les corrections choisies pour le système mais également enrichir la base d'apprentissage.

3.3.2 Expérimentation

3.3.2.1 Questions de recherche et hypothèses des expérimentations

Le fait d'aborder cette partie dans notre recherche nous permettra de répondre à la question de recherche suivante :

QR 7 : quel est l'impact des refactorisations sur la qualité des systèmes corrigés ? Pour traiter cette question, nous utiliserons le modèle de qualité développé par Khomh *et al.* [22], PQMOD, dans leur méthode DEQUALITE, pour nous assurer que nous avons amélioré la qualité du système grâce aux corrections. Ainsi, nous calculerons les facteurs de qualité avant et après la correction, nous en déduirons ensuite

l'impact des refactorisations sur les systèmes corrigés. Nous pourrons ensuite reprendre le processus jusqu'à atteindre une stabilité.

3.3.2.2 Sujets des expérimentations

Nous utilisons ici les mêmes sujets (les quatre défauts) corrigés par FCAReCor vu que l'objet de nos experimentations est de déterminer l'impact de ces corrections sur la qualité du système.

3.3.2.3 Objets et processus des expérimentations

Nous utilisons ici également les 10 systèmes libres Java utilisés précédemment. Aussi, mesurerons-nous la qualité du système avec les défauts de conception (avant la correction) puis, nous mesurerons une seconde fois ces attributs de qualité après la correction.

Nous analyserons par la suite les différences dans les mesures pour conclure sur l'impact des corrections sur la qualité du système.

CHAPITRE 4

PLAN DE RECHERCHE

Dans ce chapitre, nous présentons d'abord un résumé du travail réalisé, puis nous discutons les possibles directions de recherche et enfin nous terminons par une présentation du plan de publications possibles de nos résultats.

4.1 État actuel de notre travail de recherche

Nous avons réalisé les étapes suivantes dans le cadre de notre travail de recherche :

- un état de l'art de la recherche sur les approches pour la détection des défauts de conception ;
- un état de l'art de la recherche sur les techniques de correction pour la détection des défauts ;
- un état de l'art de la recherche sur les algorithmes d'apprentissage supervisé, les systèmes de recommandation, l'application de l'AFC et l'ARC pour la catégorisation ;
- une étude empirique sur la possibilité d'appliquer les SVM pour la détection des défauts. Cette phase nous a permis de constater, à travers les résultats de détection dans le tableau 4.I, que les SVM sont une bonne alternative pour la détection automatique des défauts de conception.

Ces étapes ont contribué à :

- montrer qu'il y a beaucoup de limitations dans les approches utilisées pour la détection des défauts de conception dont certaines étapes importantes restent encore manuelles ;

	ArgoUML v0.18.1	Xerces v2.7.0
Blob	84% 91%	94% 97%
Spaghetti Code	73% 89%	66% 60%

Tableau 4.I – Résultats de l'application de SVM pour la detection

(Dans chaque ligne, la première ligne indique la précision et la deuxième le rappel, nous avons utilisé le "10 cross validation")

- montrer que les SVM sont une alternative pour la détection automatique des défauts de conception ;
- considérer qu'un système de recommandation serait pertinent pour la suggestion des refactorisations pour la correction ;
- montrer que l'AFC et l'ARC sont des directions prometteuses pour la catégorisation et l'identification des parties à restructurer dans les systèmes ;
- proposer un cadre qui prendra en compte la détection, la correction automatique et l'impact sur la qualité.

4.2 Plan de nos travaux futurs

Afin d'améliorer les résultats des SVM, nous avons initié les travaux suivants qui sont actuellemnt en cours :

- personnalisation de notre version de SVM, SVMDetect, afin de prendre en compte les métriques spécifiques à chaque défaut de conception et d'être capable d'extraire les attributs les plus importants dans la détection de chaque défaut ;
- implantation de la prise en compte des retours usagers pour SVMDetect ;

Ces travaux permettront d'adresser les questions de recherche **QR1**, **QR2**, **QR3**.

Nous allons bientôt explorer la définition de règles pour le parcours automatique des treillis en fonction de chaque défaut. Pour cela, nous reprenons l'application de Moha *et*

al. [30] basée sur l'AFC et l'ARC pour le blob afin de raffiner les règles de parcours du treillis pour le blob puis nous étendrons les résultats aux autres défauts de conception.

Nous analyserons également l'intégration de ces règles et les caractéristiques des refactorisations dans notre système de recommandation. Ces travaux nous permettront d'adresser les questions de recherche **QR4**, **QR5**, **QR6**.

Pour finir, nous envisageons étudier l'application du modèle de qualité PQMOD pour la mesure de l'impact de la correction afin de répondre à la question de recherche **QR7**.

4.3 Échéancier et plan de publications

La figure 4.1 et le tableau 4.II présentent notre plan de recherche ainsi que nos principales phases de publications. Nous planifions de publier les résultats de nos travaux futurs dans les conférences et les journaux cités dans le tableau.

Session	Date	Conférence	Contribution
A10	10 - 2010	**CSMR'10** : Conference on Software Maintenance and Reengineering	L'application de SVMDetect pour la détection des défauts de conception
H11	02 - 2011	**ICPC'11** : Conf. Internationale sur la compréhension de programmes	La prise en compte des retours d'usagers pour l'amélioration de la détection
H11	04 - 2011	**ICSM'11** : Conf. Internationale sur la maintenance de programmes	L'application de l'AFC pour la suggestion des restructurations
E11	07 - 2011	**WCRE'11** : Conf. Internationale sur la ré-ingénierie des programmes	FCAReCor, Système de recommandation pour la correction automatique
A11	12 - 2012	**Journal IST** : *Information and Software Technology*	Impact de la correction sur la qualité des systèmes
H12	03 - 2012		Rédaction de la thèse
E12	07 - 2012	**WCRE'12** : Conf. Internationale sur la ré-ingénierie des programmes	Comparaison des outils de détection et de correction des défauts
A12	11 - 2012	**Journal TSE** : *IEEE Transactions of Software Engineering*	SUDERCO, un cadre complet pour la détection et la correction des défauts
A12	12 - 2012		Dépôt de la thèse

Tableau 4.II – Plan de publications.

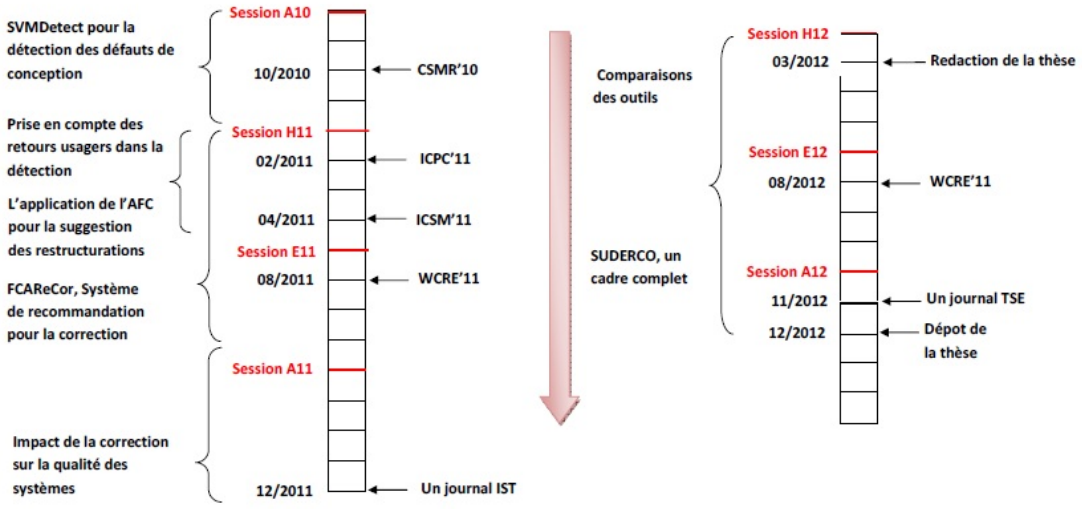

Figure 4.1 – Plan de recherche présent et futur

BIBLIOGRAPHIE

[1] Gediminas Adomavicius and YoungOk Kwon. New recommendation techniques for multi-criteria rating systems. *Intelligent Systems, vol. 22, no. 3*, 2007.

[2] Gediminas Adomavicius, Ramesh Sankaranarayanan, Shahana Sen, and Alexander Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.*, 23(1) :103–145, 2005.

[3] Gediminas Adomavicius and Alexander Tuzhilin. Expert-driven validation of rule-based user models in personalization applications. *Data Min. Knowl. Discov.*, 5(1-2) :33–58, 2001.

[4] Gediminas Adomavicius and Er Tuzhilin. Toward the next generation of recommender systems : A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17 :734–749, 2005.

[5] El Hachemi Alikacem and Houari A. Sahraoui. Détection d'anomalies utilisant un langage de règle de qualité. In *LMO*, pages 185–200. Hermès Science Publications, 2006.

[6] Lowell Jay Arthur. *Software Evolution : The Software Maintenance Challenge*. Wiley-Interscience, New York, NY, USA, 1988. ISBN : 0-471-62871-9, 1988.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[8] Jr. Frederick P. Brooks. *The Mythical Man-Month : Essays on Software Engineering*. Addison-Wesley Publishing Company, Reading, MA , USA, 1975. ISBN : 0201835959., 1975.

[9] William J. Brown, Raphael C. Malveau, William H. Brown, Hays W. McCormick

III, and Thomas J. Mowbray. *Anti Patterns : Refactoring Software, Architectures, and Projects in Crisis*. John Wiley and Sons, 1st edition, March 1998.

[10] Corinna Cortes and V. Vapnik. Support-vector networks, machine learning. 1995.

[11] Karim Dhambri, Houari Sahraoui, and Pierre Poulin. Visual detection of design anomalies. In *CSMR '08 : Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering*, pages 279–283, Washington, DC, USA, 2008. IEEE Computer Society.

[12] Kai-Bo Duan and S. Sathiya Keerthi. Which is the best multiclass svm method ? an empirical study. In Nikunj C. Oza, Robi Polikar, Josef Kittler, and Fabio Roli, editors, *Multiple Classifier Systems*, volume 3541 of *Lecture Notes in Computer Science*, pages 278–285. Springer Berlin / Heidelberg, 2005.

[13] A. Cornuéjols et L. Miclet. *Apprentissage Artificiel : Méthodes et Algorithmes*. Eyrolles, 2002.

[14] John Foster. *Cost Factors in Software Maintenance*. Phd thesis, Computer Science Department, University of Durham, 1993.

[15] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring : Improving the Design of Existing Code*. Addison-Wesley Professional, 1999.

[16] Robert Godin and Hafedh Mili. Building and maintaining analysis-level class hierarchies using galois lattices. In *OOPSLA '93 : Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, pages 394–410, New York, NY, USA, 1993. ACM.

[17] T. Hastie and R. Tibshirani. Classification by pairwise coupling. in : Jordan, m.i., kearns, m.j., solla, a.s. (eds.). *Advances in Neural Information Processing Systems 10*, 1998.

[18] Marianne Huchard. Analyse formelle de concepts pour l'ingénierie des modèles. Notes de cours, 2008.

[19] Marianne Huchard, Mohamed Rouane Hacene, Cyril Roume, and Petko Valtchev. Relational concept discovery in structured datasets. *Ann. Math. Artif. Intell.*, 49(1-4) :39–76, 2007.

[20] Nello Cristianini John Shawe-Taylor. Support vector machines and other kernel-based learning methods. *Cambridge University Press*, 2000.

[21] Foutse Khomh. *Patrons et qualité des programmes orienté objets*. PhD thesis, Université de Montréal, September 2010.

[22] Foutse Khomh and Yann-Gaël Guéhéneuc. Dequalite : building design-based software quality models. In *PLoP '08 : Proceedings of the 15th Conference on Pattern Languages of Programs*, pages 1–7, New York, NY, USA, 2008. ACM.

[23] Douglas Kirk, Marc Roper, and Neil Walkinshaw. Using attribute slicing to refactor large classes. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2006.

[24] Edward J. Klimas, Suzanne Skublics, and David A. Thomas. *Smalltalk with Style*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[25] Bennett P. Lientz and E. Burton Swanson. *Software Maintenance Management*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1980. ISBN : 0201042053, 1980.

[26] Rujie Liu, Yuehong Wang, Takayuki Baba, Daiki Masumoto, and Shigemi Nagata. Svm-based active feedback in image retrieval using clustering and unlabeled data. *Pattern Recogn.*, 41(8) :2645–2655, 2008.

[27] Radu Marinescu. Detection strategies : Metrics-based rules for detecting design flaws. In *In Proceedings of the IEEE 20th International Conference on Software Maintenance*, pages 350–359. IEEE Computer Society Press, 2004.

[28] T. Mens and T. Tourwe. A survey of software refactoring. *Software Engineering, IEEE Transactions on*, 30(2) :126 – 139, feb. 2004.

[29] Naouel Moha. *DECOR : détection et correction des défauts dans les systèmes orientés objet*. PhD thesis, Université de Montréal et Université de Lille, August 2008.

[30] Naouel Moha, Yann-Gael Gueheneuc, Laurence Duchien, and Anne-Francoise Le Meur. Decor : A method for the specification and detection of code and design smells. *IEEE Transactions on Software Engineering*, 36 :20–36, 2010.

[31] Naouel Moha, Yann-Gael Gueheneuc, and Pierre Leduc. Automatic generation of detection algorithms for design defects. *Automated Software Engineering, International Conference on*, 0 :297–300, 2006.

[32] R. J. Mooney. Content-based book recommending using learning for text categorization. *In ACM SIGIR'99, Workshop on Recommender Systems : Algorithms and Evaluation*, 1999.

[33] Matthew James Munro. Product metrics for automatic identification of "bad smell" design problems in java source-code. In *METRICS '05 : Proceedings of the 11th IEEE International Software Metrics Symposium*, page 15, Washington, DC, USA, 2005. IEEE Computer Society.

[34] A. Nakamura and N. Abe. Collaborative filtering using weighted majority prediction algorithms. *In Proceedings of the 15th International Conference on Machine Learning*, 1998.

[35] William F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1992.

[36] J. Platt, N. Cristanini, and J. Shawe-Taylor. Large margin dags for multiclass classification. *Advances in Neural Information Processing Systems 12*, pages 543–557, 2000.

[37] Roger S. Pressman. *Software Engineering - A Practitioner's Approach*. McGraw-Hill Higher Education, 5th edition, November 2001. ISBN : 0-07-249668-1, 2001.

[38] Charles Rich and Richard C. Waters. *The Programmer's Apprentice*. ACM Press Frontier Series and Addison-Wesley, 1st edition, January 1990.

[39] Arthur J. Riel. *Object-Oriented Design Heuristics*. Addison-Wesley, 1996.

[40] J. J. Rocchio. Relevance feedback in information retrieval. *In : Salton : The SMART Retrieval System : Experiments in Automatic Document Processing, Chapter 14*, 1971.

[41] Houari A. Sahraoui, Hakim Lounis, Walcélio Melo, and Hafedh Mili. A concept formation based approach to object identification in procedural code. *Automated Software Engg.*, 6(4) :387–410, 1999.

[42] Gregor Snelting and Frank Tip. Understanding class hierarchies using concept analysis. *ACM Trans. Program. Lang. Syst.*, 22(3) :540–582, 2000.

[43] Simon Tong and Edward Chang. Support vector machine active learning for image retrieval. In *MULTIMEDIA '01 : Proceedings of the ninth ACM international conference on Multimedia*, pages 107–118, New York, NY, USA, 2001. ACM.

[44] Guilherme Travassos, Forrest Shull, Michael Fredericks, and Victor R. Basili. Detecting defects in object-oriented designs : using reading techniques to increase software quality. In *OOPSLA '99 : Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 47–56, New York, NY, USA, 1999. ACM.

[45] Adrian Trifu and Iulian Dragos. Strategy-based elimination of design flaws in object-oriented systems. In *Proceedings of the 4th international Workshop on Object-Oriented Reengineering. Universiteit Antwerpen*. In Serge Demeyer, St´ephane Ducasse, and Kim Mens, editors, 2003.

[46] Nicolas Turenne. Méthode des k plus proches voisins. *INRA*, 2006.

[47] Rebecca Wirfs-Brock and Alan McKean. *Object Design : Roles, Responsibilities, and Collaborations*. Addison-Wesley, January 2003.

L'École Polytechnique se spécialise dans la formation d'ingénieurs et la recherche en ingénierie depuis 1873

École Polytechnique de Montréal

École affiliée à l'Université de Montréal

Campus de l'Université de Montréal
C.P. 6079, succ. Centre-ville
Montréal (Québec)
Canada H3C 3A7

www.polymtl.ca

ÉCOLE
POLYTECHNIQUE
MONTRÉAL